Amendments to Claims

This listing of claims will replace all prior versions and listings of claims in the application:

Listing of Claims

1. (currently amended) A method for executing a dataflow application comprising:

providing a dataflow application comprising a plurality of map components and data ports, a number of map components being linked between data ports and some map components comprising composite components having a plurality of processes;

allocating a processing thread to each ~~composite~~ map component; and

executing each ~~composite~~ map component on a separate thread.

2. (original) The method of claim 1, including assigning runtime properties to the prepared map prior to execution.

3. (original) The method of claim 1, certain map components including input ports and output ports and the executing step including reading data tokens from respective input ports and writing data tokens to respective output ports.

4. (original) The method of claim 1, including choosing the port implementations to a specific data type prior to said executing step.

5. (original) The method of claim 1, some of the ports being hierarchical comprising a plurality of hierarchical ports and the data being partitioned for parallel processing.

6. (original) The method of claim 1, including transporting data tokens among map processes.

7. (original) The method of claim 6, batching the data tokens to regulate the length of time a map process may execute without synchronization.

8. (original) The method of claim 1, including ports associated with each map process for representing and transporting multi-state null value tokens.

9. (original) The method of claim 8, the null value tokens including an error null.

10. (original) A method of deadlock management in a multi-thread, parallel processing data management system having ports for sending and receiving data tokens comprising:

allocating at least one thread to a first process and at least one thread to a second process, wherein the first and second processes are connected through a queue via ports;

determining if a thread is blocked, waiting on another thread, and determining if the blocked thread is sending data or receiving data; and

determining if a deadlock exists by building a wait graph of blocked threads in the system and determining if the graph is cyclic, that is waiting on itself, indicating a deadlock does exist.

11. (original) The method of claim 10, blocking a receiving port when a data token is not available.

12. (original) The method of claim 10, blocking a sending port when a limit on the number of data tokens in the queue is reached.

13. (original) The method of claim 10, including building a wait graph with said blocked threads and traversing said wait graph to determine if it is cyclic.

14. (original) The method of claim 10, if a deadlock is detected, correcting the deadlock by allowing the limit of data tokens on a queue to increase.

15. (original) The method of claim 14, wherein the limit of a queue associated with a sending port is allowed to increase.

16. (original) The method of claim 14, wherein the token batch size of another queue is decreased while said limit of said queue is increasing.

17. (previously presented) A dataflow application having multiple processes executed on one or more multi-threaded central processing units ("CPUs") connected by data input and output ports wherein at least some ports are implemented to indicate multiple state null value data tokens.

18. (original) The application of claim 17, wherein the multiple state null value port operations support at least three state logic.

19. (original) The application of claim 17, wherein the multiple state null values support at least a system defined error null token.

20. (original) The application of claim 19, the system defined error null being passed to a map component for later determination of a system error condition.